Vacation Itinerary Planner

Arun Thomas

Feb 20, 2021

1. Introduction

1.1 Background

We all love to travel and visit new places. Well, we are in midst of a pandemic right now, and the urge to go on a vacation is even greater, because most of us are confined to our homes. Let's see if and how we can use out Data Science leaning to help us have a better vacation!

1.2 Problem

One of the hardest parts of planning a vacation is preparing an itinerary. We all know that a bad itinerary can completely ruin our vacation, because we might either be forced to spend more money (money wastage) or more commonly, we might spend a lot of time unnecessarily travelling. We can gave some sample itinerary from the internet, but they are usually very general and wouldn't take into account your personal interests or how much time we have in hand.

1.3 Interest

Our interest is in using data science and python to create a personalized vacation itinerary planner, to which we can input the place, time available for travel and preferred venues, and it returns a detailed itinerary where we have a list of places to travel to , and the dates when we should travel.

2. Data acquisition and cleaning

2.1 Data sources

Our primary data source is the FourSquare API, we get all the venue information from there. We can also use the 'export to kml file' of the google maps feature, to input the kml file with the pinpoints on all the destinations we want to travel to.

2.2 Data cleaning

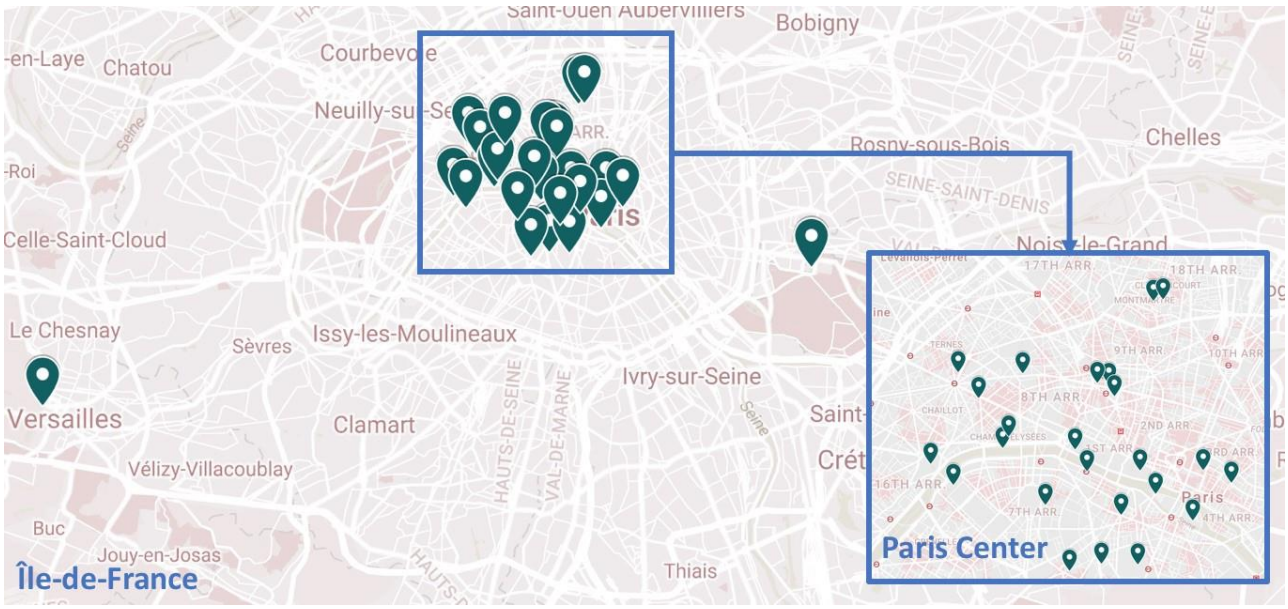
We need to transform the data that we get from the FourSquare response API. We have already seen how to do that in our previous lectures and I have done the same in this project too. So basically, da downloaded or scraped from multiple sources were combined into one table.

I also checked for outliers in the data. I found there were some extreme outliers, mostly caused by some types of small sample size problem.

2.3 Feature selection

The main features which are important to us are, the Latitude, Longitude, and the venue name.

3. Exploratory Data Analysis



First, I had to gather the Google map pins geo-locations in a 2D format (something that could be stored in a numpy array). Translating those pins into[longitude, latitude] would be perfect.

Since this was a one-off, I looked for a quick way to extract this info from the Google map. This StackOverflow query gave me all that I needed.

Basically, one needs to go to *google.com/maps/d/kml?mid={map_id}* and download a *.kmz file. Then, manually change the extension of the *.kmz file to a *.zip file. Extract the file, and open *doc.kml* in a text editor of your choice (SublimeText is my personal go-to).

Then, you may decide to manually CTRL+F to search for <coordinates> fields, or decide to *not* be so lazy about it and use BeautifulSoup (as shown below)!

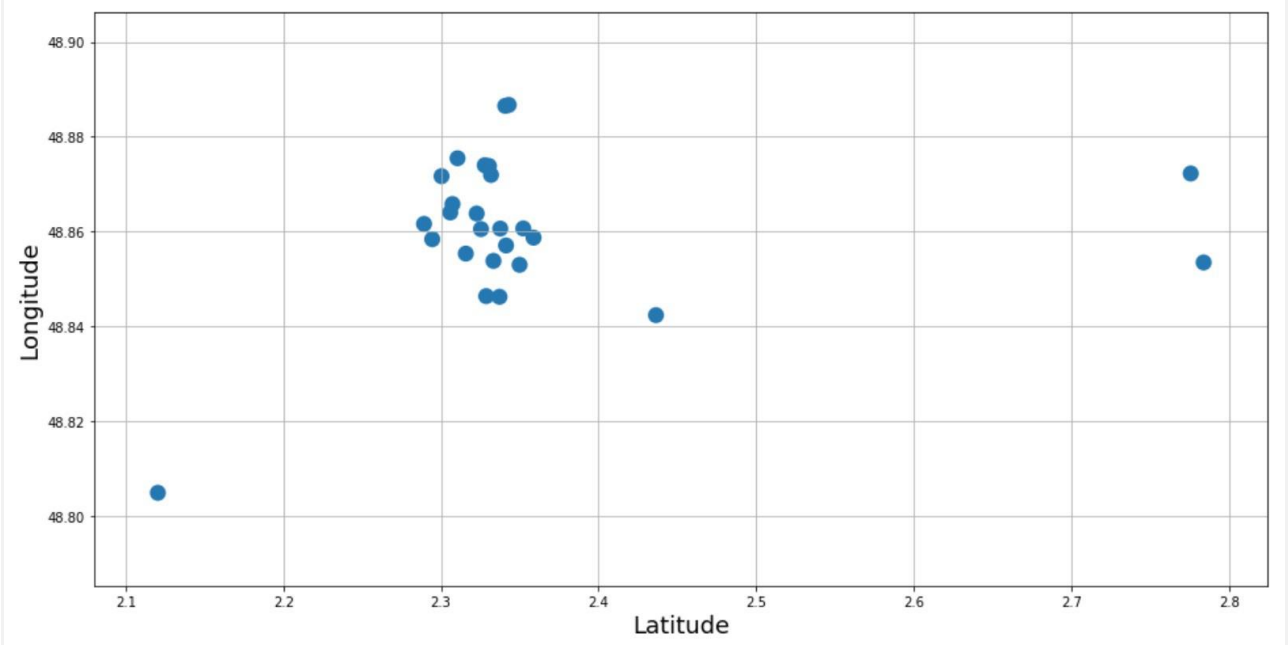
Once I extracted the coordinates from the XML file, I stored the coordinates in a dataframe. The total number of landmarks is 26 (stored in <*Placemark*></*Placemark*> in the XML file).

Out[5]:

	Longitude	Latitude	Landmark
0	2.328729	48.846361	French Alliance Paris Ile-De-France
1	2.289282	48.861596	Trocadéro Gardens
2	2.294483	48.858370	Eiffel Tower
3	2.340802	48.886503	Place du Tertre
4	2.343023	48.886706	The Basilica of the Sacred Heart of Paris
5	2.300375	48.871669	Louis Vuitton Maison Champs Élysées
6	2.330479	48.873820	Galeries Lafayette

Dataframe populated with info from the XML file (first 7 rows only shown)

From the dataframe which stored coordinates and the name of the landmark/sight, I generated a scatter plot.



Google Map Pins Scatter Plot

K-Means Clustering

In general, unsupervised learning methods are useful for datasets without labels AND when we do not necessarily know the outcome we are trying the predict. These algorithms typically take one of two forms: (1) clustering algorithms, or (2) dimensionality reduction algorithms.

In this section, we'll focus on *k*-means, which is a clustering algorithm. With *k*-means, a pre-determined number of clusters is provided as input and the algorithm generates the clusters within the un-labeled dataset.

k-means generates a set of *k* cluster centroids and a labeling of input array *X* that assigns each of the points in *X* to a unique cluster. The algorithm determines cluster centroids as the arithmetic mean of all the points belonging to the cluster, AND defines clusters such that each point in the dataset is closer to its own cluster center than to other cluster centers.

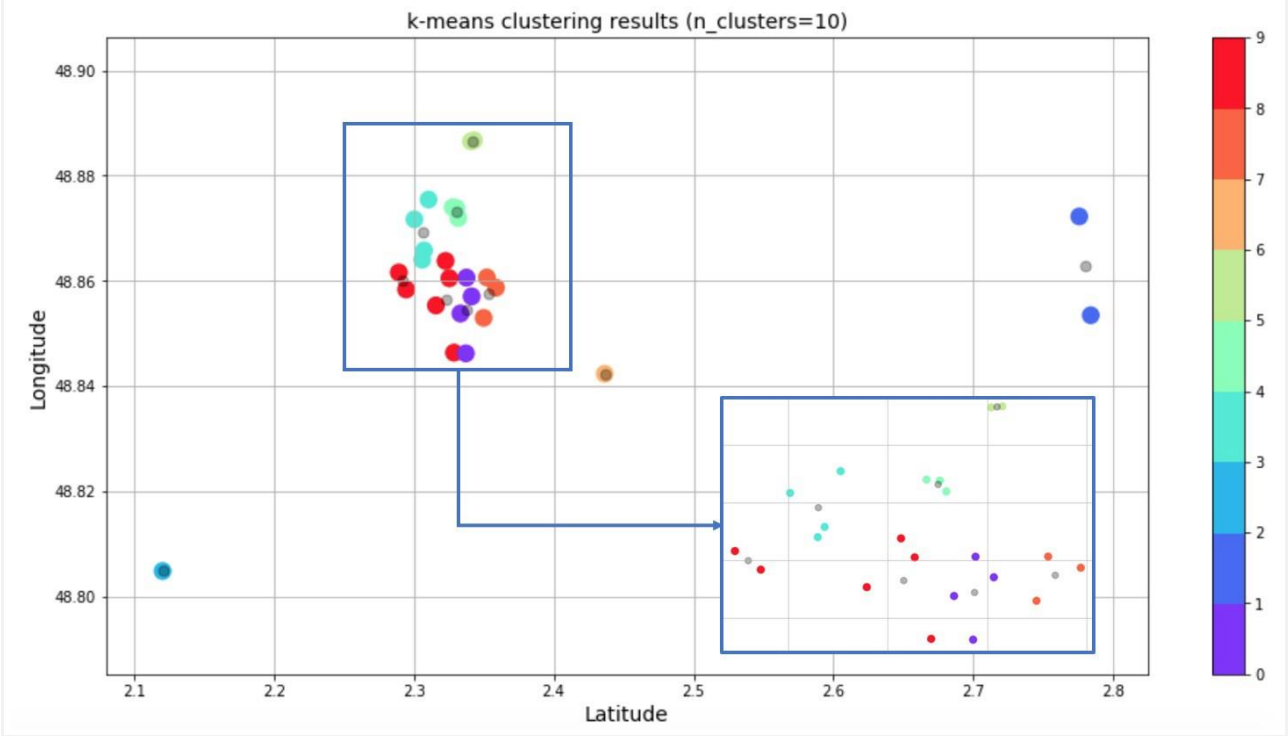
The mathematical condition for the K clusters C_k and the K centroids μ_k can be expressed as:

$$\text{Minimize} \sum_{k=1}^{K} \sum_{x_n \in C_k} ||x_n - \mu_k||^2 \text{ with respect to } C_k, \mu_k.$$

source: https://bit.ly/1z29ZIV

It can be implemented in Python using sklearn as follows:

As one can see in the scatter plot below, I generated 10 clusters — one for each vacation day. But the sights in the center of Paris are so close to each other, it is quite difficult to distinguish one cluster from the other. The resulting predictions were also sorted and stored in a dataframe.



k-means was used to generate 10 clusters (one for each day), the black dots represent cluster centers

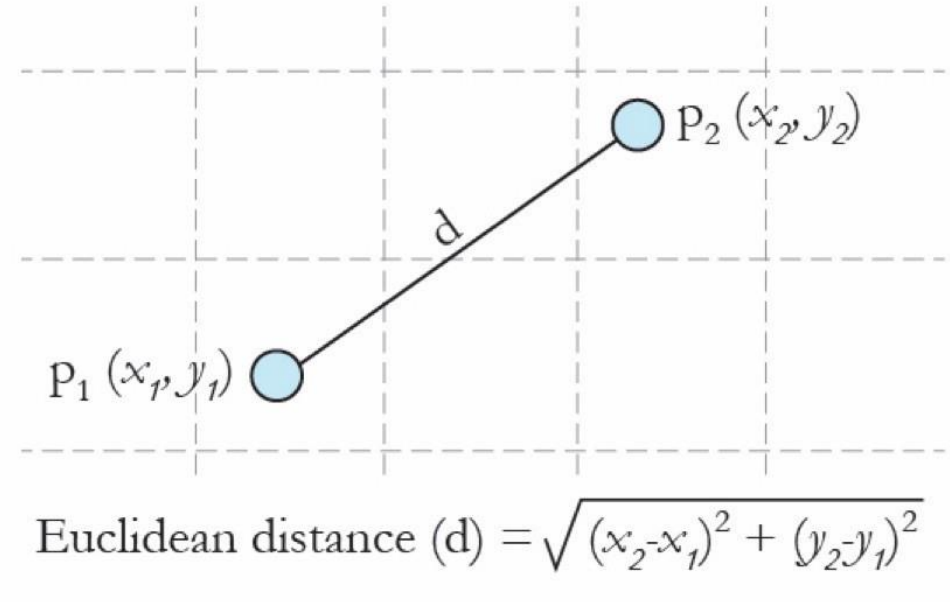
Using the 10 clusters generated by *k*-means, I generated a dataframe that assigns a day of the week to each cluster. This would constitute an example of a schedule.

	Longitude	Latitude	Landmark	Cluster	Vacation Day
1	2.289282	48.861596	Trocadéro Gardens	0	Monday
2	2.294483	48.858370	Eiffel Tower	0	Monday
12	2.775808	48.872234	Disneyland Paris	1	Tuesday
17	2.784017	48.853465	La Vallée Village	1	Tuesday
22	2.358804	48.858703	Le Marais	2	Wednesday
13	2.352245	48.860642	The Centre Pompidou	2	Wednesday
14	2.349902	48.852968	Cathédrale Notre-Dame de Paris	2	Wednesday

k-means results as Vacation Days of the Week (first 3 days only shown)

Now, at this stage, I could have simply handed over the sorted dataframe, re-organised the Google map pins by layers (i.e. each layer representing one day), and that's it — itinerary complete.

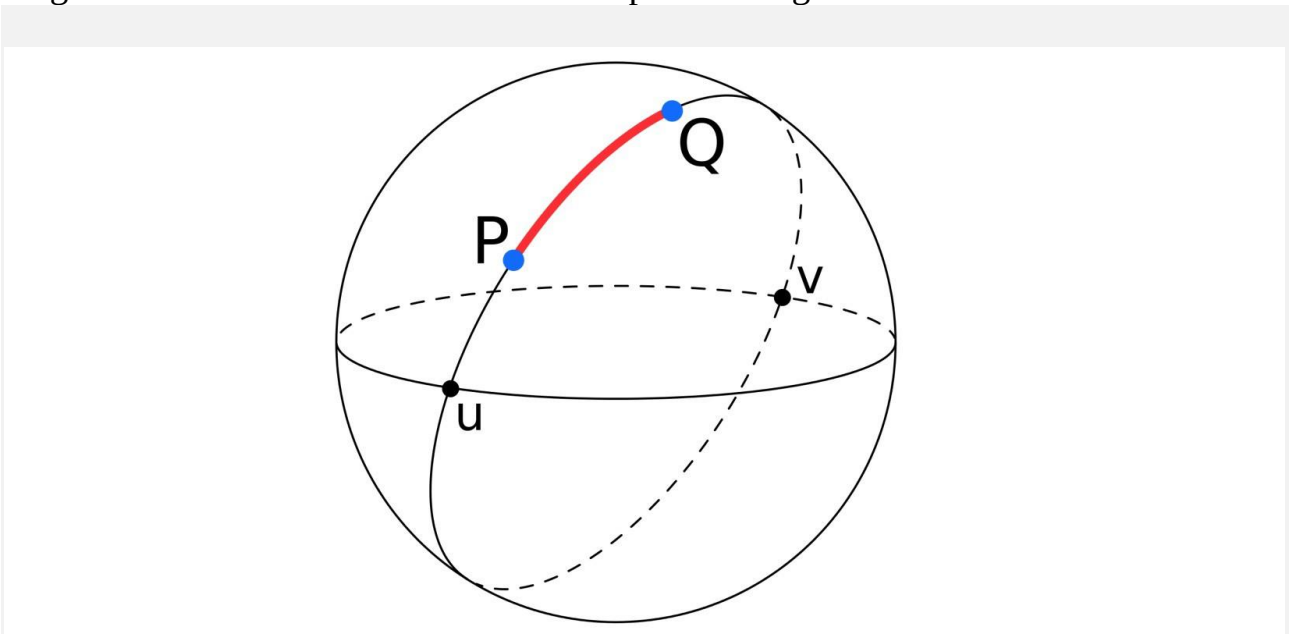
But something was still bugging me, and that is that *k*-means was generating clusters based on the Euclidean distance between points — meaning the straight-line distance between two pins in the map.



But as we know, the Earth isn't flat (right?) so I was wondering if this approximation was affecting the clusters being generated, especially since we have quite a few landmarks far from the high-density region in the center of Paris.

Because the Earth isn't flat: enter HDBSCAN

Hence, we need a clustering method that can handle Geographical distances, meaning lengths of the shortest curve between two points along the surface of the Earth.



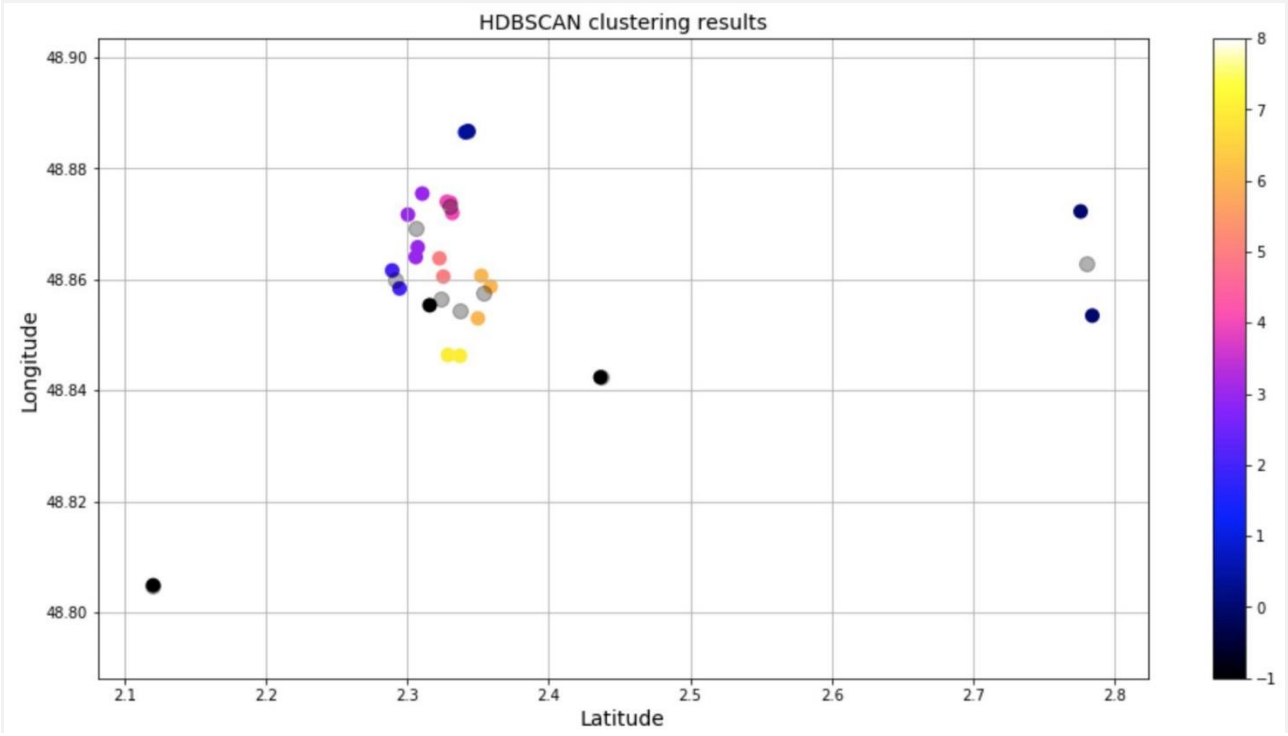
A density function such as HDBSCAN, which is based on the DBScan algorithm, may be useful for this.

Both HDBSCAN and DBSCAN algorithms are density-based spatial clustering method that group together points that are close to each other based on a distance measurement and a minimum number of points. It also marks as outliers the points that are in low-density regions.

Thankfully, HDBSCAN supports haversine distance (i.e. longitude/latitude distances) which will properly compute distances between geo-locations. For more on HDBSCAN, check out this blog post.

HDBSCAN isn't included in your typical Python distribution so you'll have to *pip* or *conda* install it. I did so, and then ran the code below.

And ended up with the following scatter plot and dataframe. We see that isolated points were clustered in cluster '-1' which means they were identified as 'noise'.



HDBSCAN generated 9 clusters and 3 data points as 'noise'

[55]:

	Longitude	Latitude	Landmark	Cluster	Vacation Day
23	2.315835	48.855307	Rodin Museum	-1	NA/Noise
20	2.436470	48.842348	Sainte-Chapelle de Vincennes	-1	NA/Noise
10	2.120355	48.804865	Palace of Versailles	-1	NA/Noise
12	2.783593	48.867386	Disneyland Paris	0	Monday
17	2.783418	48.852891	La Vallee Village	0	Monday
3	2.340802	48.886503	Place du Tertre	1	Tuesday
4	2.343030	48.886711	The Basilica of the Sacred Heart of Paris	1	Tuesday
1	2.289282	48.861596	TrocadÃ©ro Gardens	2	Wednesday
2	2.293046	48.856093	Eiffel Tower	2	Wednesday
18	2.307796	48.866229	Champs-Ã‰lysÃ©es	3	Thursday
11	2.305937	48.864011	Bateaux-Mouches	3	Thursday
25	2.310732	48.875026	MusÃ©e Jacquemart-AndrÃ©	3	Thursday
5	2.300457	48.871545	Louis Vuitton Maison Champs Ã‰lysÃ©es	3	Thursday
8	2.328071	48.874010	Printemps Haussmann	4	Friday
7	2.331601	48.871970	Palais Garnier	4	Friday
6	2.331978	48.873734	Galeries Lafayette Haussmann	4	Friday
19	2.325354	48.860547	MusÃ©e d'Orsay	5	Saturday
21	2.322672	48.863788	MusÃ©e de l'Orangerie	5	Saturday
24	2.333328	48.853798	Saint-Germain-des-PrÃ©s	6	Sunday
15	2.337160	48.846222	Luxembourg Gardens	6	Sunday
16	2.341325	48.857050	Pont Neuf	6	Sunday
9	2.337644	48.860611	Louvre Museum	6	Sunday
0	2.328528	48.846271	French Alliance Paris Ile-De-France	6	Sunday

HDBSCAN results as Vacation Days of the Week (first 3 days only shown)

Unsurprisingly, we end up with several points being flagged as noise. Since the minimum number of points for a HDBSCAN cluster is 2, isolated locations like the *Palais de Versailles* were categorised as noise. *Sainte-Chapelle de Vincennes*, and *Musée Rodin* suffered a similar fate.

The interesting part however is in the number of clusters that HDBSCAN identified which is 9 — one less than the set vacation days. I guess that for the number of sights/data points that we chose, 10 days sounds like it'll be okay.

Ultimately, the results from *k*-means were the ones we used to lay out a schedule as the clusters generated by *k*-means were similar to the ones generated by HDBSCAN and all data points were included.

The clustering method presented here can be improved of course. One possible improvement is in the addition of a weight feature for the datapoints. For instance, the weights could represent the amount of time needed to fully visit a particular venue (e.g. *Le Louvre* easily takes one full day to appreciate), and hence this would affect total number of data points in a cluster with highly weighted points — something to investigate in future projects.

4. Conclusions

In this study, I created a vacation itinerary planner that prepares a customized vacation itinerary for the user, after taking the no. of days of vacation, the interested venues (in the given place).Here I did for 25 venues in Paris for 7 days.

[55]:

	Longitude	Latitude	Landmark	Cluster	Vacation Day
23	2.315835	48.855307	Rodin Museum	-1	NA/Noise
20	2.436470	48.842348	Sainte-Chapelle de Vincennes	-1	NA/Noise
10	2.120355	48.804865	Palace of Versailles	-1	NA/Noise
12	2.783593	48.867386	Disneyland Paris	0	Monday
17	2.783418	48.852891	La Vallee Village	0	Monday
3	2.340802	48.886503	Place du Tertre	1	Tuesday
4	2.343030	48.886711	The Basilica of the Sacred Heart of Paris	1	Tuesday
1	2.289282	48.861596	TrocadÃ©ro Gardens	2	Wednesday
2	2.293046	48.856093	Eiffel Tower	2	Wednesday
18	2.307796	48.866229	Champs-Ã‰lysÃ©es	3	Thursday
11	2.305937	48.864011	Bateaux-Mouches	3	Thursday
25	2.310732	48.875026	MusÃ©e Jacquemart-AndrÃ©	3	Thursday
5	2.300457	48.871545	Louis Vuitton Maison Champs Ã‰lysÃ©es	3	Thursday
8	2.328071	48.874010	Printemps Haussmann	4	Friday
7	2.331601	48.871970	Palais Garnier	4	Friday
6	2.331978	48.873734	Galeries Lafayette Haussmann	4	Friday
19	2.325354	48.860547	MusÃ©e d'Orsay	5	Saturday
21	2.322672	48.863788	MusÃ©e de l'Orangerie	5	Saturday
24	2.333328	48.853798	Saint-Germain-des-PrÃ©s	6	Sunday
15	2.337160	48.846222	Luxembourg Gardens	6	Sunday
16	2.341325	48.857050	Pont Neuf	6	Sunday
9	2.337644	48.860611	Louvre Museum	6	Sunday
0	2.328528	48.846271	French Alliance Paris Ile-De-France	6	Sunday

5. Future directions

There are improvements we can do with our project, we can for example, give more customization options in the venue selection details, factor in the mode of transport and cost of accommodation.